# PHIDGET SBC
## (SINGLE BOARD COMPUTER)

## Table des matières

**Présentation....3**
Système embarqué....3
La carte Phidget SBC 1070....3
Caractéristiques....4
**L'architecture ARM....5**
Présentation....5
Architecture RISC....5
Pipeline....5
Le coeur ARM9....5
Intérêts de la carte Phidget SBC (Single Board Computer)....6
Mémoire flash NAND....7
Procédure de démarrage....8
**L'environnement de développement....9**
Mise en situation....9
Compilation croisée....9
Chaîne de compilation traditionnelle sous Linux....9
Buildroot....10
**Manipulations basiques....11**
Partie A : manuel d'installation....11
Partie B : mise en oeuvre de la communication réseau....12
Partie C : fabrication d'un exécutable pour la carte TINI....14
Partie D : déploiement et exécution d'une application....17
**Annexe 1 : Obtenir des informations sur le système embarqué....20**
Exemple de fiche de configuration de la carte....21
**Annexe 2 : Glossaire logiciel....22**
**Annexe 3 : Exemple de tests de validation....23**
**Annexe 4 : Pré-installation sur le poste de développement....24**
**Annexe 5 : Données persistantes, Journalisation (log)....26**
**Annexe 6 : Screenshot de menuconfig....27**
**Annexe 7 : Principales options de configuration....30**
**Annexe 8 : Chien de garde (watchdog)....31**
Le Watchdog est-il activé sur la phidget ?....31
Comment activer le Watchdog sur la phidget ?....32

**Lien : http://www.phidgets.com/**

# Présentation

## Système embarqué

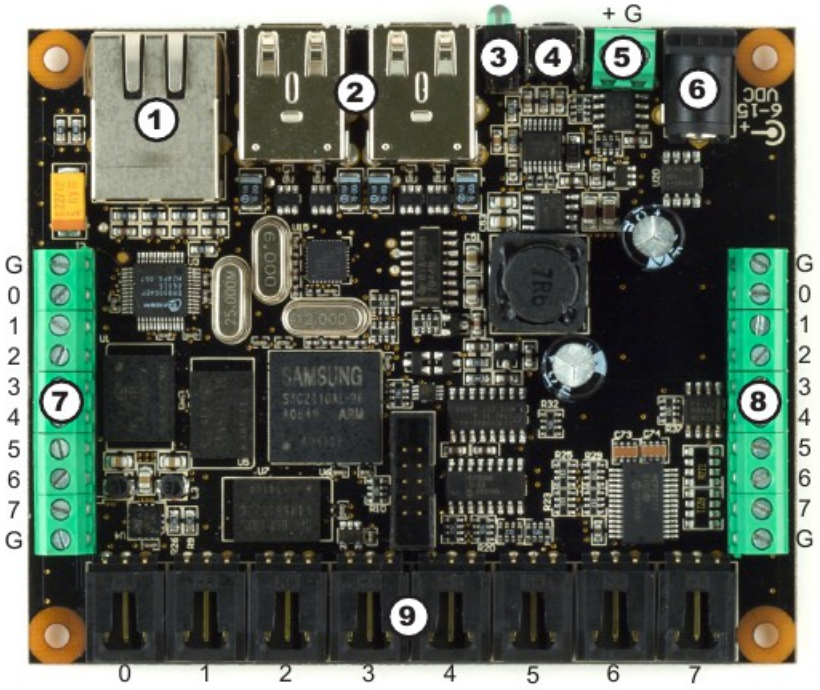
*Un système embarqué (ou système enfoui) peut être défini comme un système électronique et informatique autonome, qui est dédié à une tâche bien précise. Ses ressources disponibles sont généralement limitées (coût faible, taille réduite, consommation restreinte, espace mémoire limité de l'ordre de quelques Mo maximum, puissance de calcul juste nécessaire pour répondre aux besoins, ...).*

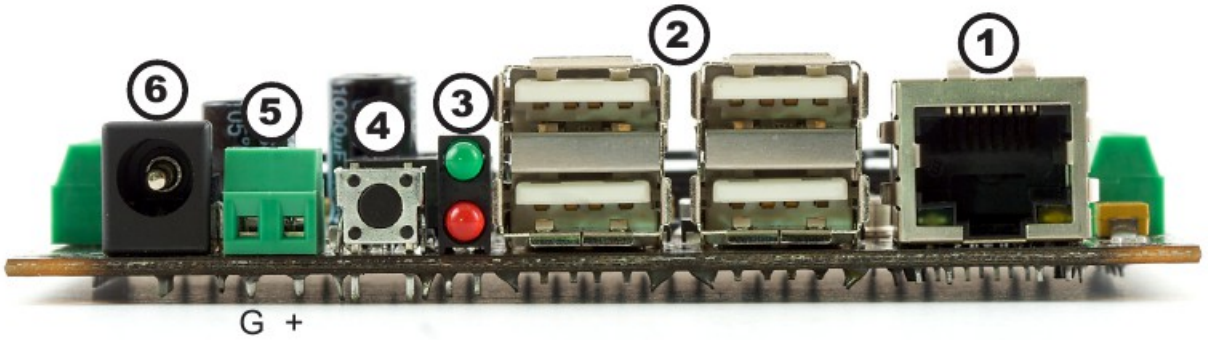
## La carte Phidget SBC 1070

La carte Phidget (SBC 1070) est un système embarqué basé sur un circuit S3C2410A de Samsung. Ce processeur RISC 16/32 bits possède un coeur ARM9.

Numbered in the circles on the diagram:

1. 10/100baseT Ethernet
2. Four USB Full-Speed Ports
3. Indicator LEDs
4. Reboot / Reset Button
5. Power input terminal
6. Power input jack
7. Eight Interface Kit Digital Inputs (Indexed 0 to 7)
8. Eight Interface Kit Digital Outputs (Indexed 0 to 7)
9. Eight Interface Kit Analog Inputs (Indexed 0 to 7)



## *Caractéristiques*

Processeur : SC32410 (coeur ARM-920T de Samsung) - 266 MHz (max)
Mémoire flash NAND : 64 Mo
SDRAM : 64 Mo
Contrôleur Ethernet : DM9000
Ethernet : 10/100BASET (MDI automatique)
2 USB full-speed ports
Hub USB : 4 ports
Interface Kit 8/8/8 intégré : 8 entrées analogiques, 8 entrées logiques et 8 sorties logiques

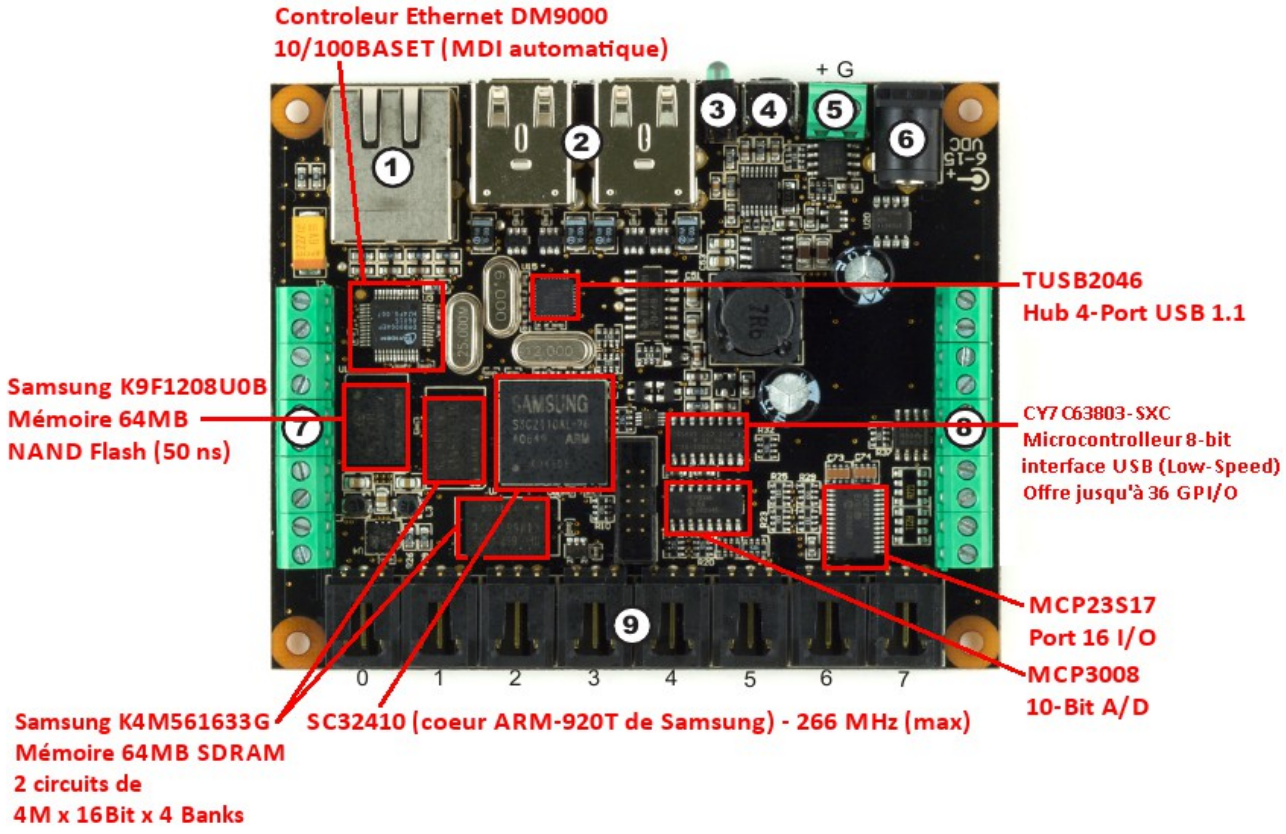
Alimentation : 12 Vcc/2 A via adaptateur secteur.
Consommation maxi : 487 mA.

Entrées analogiques :
- conversion A/D 10 bits
- impédance : 900 KOhms
- consommation totale : limitée à 400 mA

Entrées logiques :
- résistance de pull-up : 15 KOhms
- temps de détection mini : 4 ms

Sorties logiques :
- impédance : 250 Ohms
- compatibles avec entrées TTL ou CMOS

Dimensions : 98 x 81 x 20 mm
Prix TTC (indicatif) : 113.50 € (chez www.gotronic.fr)



# L'architecture ARM

## Présentation

Dotés d'une architecture relativement simple et bénéficiant d'une faible consommation, les processeurs ARM sont devenus très présents dans le domaine de l'informatique embarquée, en particulier la téléphonie mobile et les tablettes.

Les architectures ARM, développées par ARM Ltd, sont des architectures RISC (Reduced Instruction-Set Computer) 32 bits.

## Architecture RISC

Dans une architecture RISC, du fait que chaque instruction était simple, le décodage et l'exécution par le processeur sont très rapides, idéalement en un seul cycle d'horloge. De cette manière, il est possible de débuter une nouvelle instruction à chaque cycle d'horloge : ceci constitue le « pipeline ».

## Pipeline

Un pipeline est un élément de l'architecture d'un microprocesseur dans lequel l'exécution des instructions est découpée en étages, au rythme du signal d'horloge. Dans le cas où il faut plusieurs étapes pour l'accomplissement d'une instruction, un microprocesseur sans pipeline devra attendre la fin de la dernière étape pour commencer l'exécution d'une nouvelle instruction. Si l'on insère des registres tampons (pipeline registers) entre chaque unité à l'intérieur du processeur, celui-ci peut alors contenir plusieurs instructions, chacune à une étape différente. Une architecture superscalaire contient plusieurs pipelines en parallèle. Il est alors possible d'exécuter plusieurs instructions simultanément.

## Le coeur ARM9

ARM Ltd. a développé plusieurs coeurs dont le coeur ARM9 qui comporte 5 niveaux de pipeline. Cela permet ainsi la réduction du nombre d'opérations logiques sur chaque cycle d'horloge et donc une amélioration des performances en vitesse.

| Famille | Architecture | Coeur | Fonctionnalités | Cache (Instruction/Data) | MMU | MHz (MIPS) |
|---|---|---|---|---|---|---|
| ARM9TDMI | ARMv4T | ARM920T | 5 niveaux de pipeline, Thumb, double cache | 16 Ko/ 16 Ko | MMU avec FCSE (Fast Context Switch Extension) | 180 MHz (200 MIPS) |

En consultant les messages de démarrage de la carte phidget, on obtient les informations suivantes sur l'architecture ARM implantée :

```
# dmesg
Linux version 2.6.32.3 (root@sl) (gcc version 4.2.4) #1 PREEMPT Thu Mar 4 11:23:15
MST 2010
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
...
CPU S3C2410A (id 0x32410002)
S3C24XX Clocks, (c) 2004 Simtec Electronics
S3C2410: core 266.000 MHz, memory 133.000 MHz, peripheral 66.500 MHz
```

```
...
Memory: 64MB = 64MB total
Memory: 60560KB available (3884K code, 304K data, 124K init, 0K highmem)
...
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
s3c24xx-nand s3c2410-nand: Tacls=1, 7ns Twrph0=4 30ns, Twrph1=2 15ns
s3c24xx-nand s3c2410-nand: NAND hardware ECC
NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung NAND 64MiB 3,3V 8-bit)
...
```

*Lexique :*

- MMU (Memory Management Unit) : Gestionnaire de mémoire permettant l'adressage virtuel de la mémoire (uniquement présente sur l'ARM710 et les ARM9). La MMU est nécessaire pour faire fonctionner certains systèmes d'exploitation comme Windows CE ou la plupart des Linux (excepté ucLinux qui est un portage Linux sans MMU).

- Thumb : Codage d'instructions sur 16 bits (au lieu de 32 bits) permettant un gain de mémoire important

- Cache : Le cache contient une copie temporaire des données originelles lorsqu'elles sont coûteuses (en termes de temps d'accès) à récupérer ou à calculer par rapport au temps d'accès au cache. Une fois les données stockées dans le cache, l'utilisation future de ces données peut être réalisée en accédant à la copie en cache plutôt qu'en récupérant ou recalculant les données, ce qui abaisse le temps d'accès moyen. C'est notamment le cas entre le processeur et la mémoire vive (RAM). Dans ces cas, elle est intégrée au processeur.

- MIPS : Million d'instructions par seconde, unité de mesure des processeurs

*Remarque :*

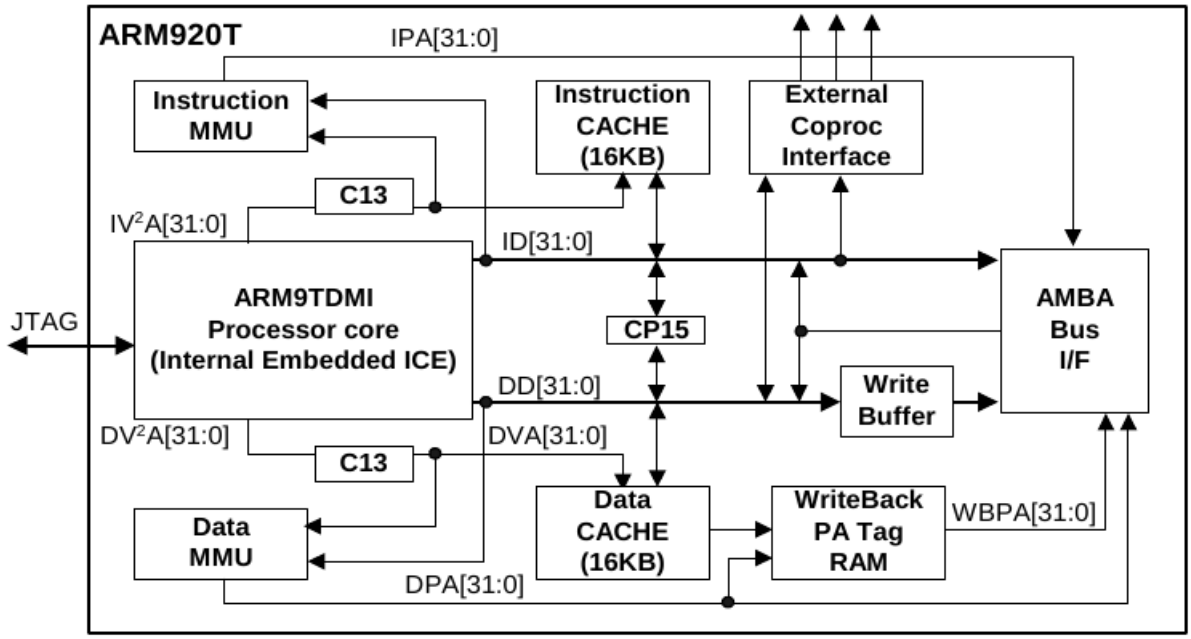
*Une particularité des processeurs ARM est leur mode de vente. En effet, ARM Ltd. ne fabrique ni ne vend ses processeurs sous forme de circuits intégrés. La société vend les licences de ses processeurs de manière à ce qu'ils soient intégrés dans le silicium par d'autres fabricants. Aujourd'hui, la plupart des grands fondeurs de puces proposent de l'architecture ARM.*

## *Intérêts de la carte Phidget SBC (Single Board Computer)*

- Un CPU 32bits à 200 MHz puissant et compact : l'ARM9 possède de nombreux périphériques intégrés (USB, contrôleur NAND, contrôleur vidéo TFT, SPI, I2C, PWM, ADC, ...) et un gestionnaire de mémoire virtuelle (MMU). Le système embarqué utilise un véritable noyau Linux 2.6. L'utilisation d'une MMU pénalise le CPU mais la fréquence de 200MHz compense allègrement l'emploie de la mémoire virtuelle.

- Économie et performance : avec ce type d'architecture ARM9/Linux, on intégre l'ensemble du coeur (CPU, mémoire, contrôleur ethernet, I/O et USB) sur une carte 98 x 81 x 20 mm pour un prix de 120 euros TTC. Tous les logiciels utilisés lors du développement sont entièrement gratuits (vim, QtCreator, gcc, gdb, ...) et le noyau Linux est gratuit et libre de droits, distribué sous licence GPL.

Sur la carte phidgetsbc, le processeur ARM est le **S3C2410A** de Samsung.




Le processeur S3C2410A supporte le démarrage (boot) à partir d'une mémoire flash de type NAND (en utilisant un buffer interne de 4Ko). Après le démarrage, le système utilise la mémoire flash de type NAND comme mémoire de masse. La carte PhidgetSBC dispose de 64 Mio de mémoire flash NAND.

## *Mémoire flash NAND*

La mémoire flash est une mémoire de masse à semi-conducteurs ré-inscriptible, c'est-à-dire une mémoire possédant les caractéristiques d'une mémoire vive mais dont les données ne disparaissent pas lors d'une mise hors tension.

Sa vitesse élevée, sa durée de vie et sa faible consommation (qui est même nulle au repos) la rendent très utile pour de nombreuses applications en informatique embarquée. De plus, ce type de mémoire ne possède pas d'éléments mécaniques, ce qui lui confère une grande résistance aux chocs.

La technique flash se décline sous deux principales formes : flash NOR (1988) et NAND (1989), d'après le type de porte logique utilisée pour chaque cellule de stockage.

La flash NAND est plus rapide à l'effacement et à l'écriture, offre une plus grande densité et un coût moins important par bit. Elle est principalement utilisée comme mémoire de masse (Carte MMC, Carte SD et Carte MS). Toutefois son interface d'entrée / sortie n'autorise que l'accès séquentiel aux données.

La flash NOR est adaptée à l'enregistrement de données informatiques destinées à être exécutées directement à partir de cette mémoire. Cette caractéristique est appelée XIP (eXecute In Place). De fait, la quasi totalité des OS des systèmes embarqués sont stockés dans une mémoire NOR.

La mémoire Flash NAND a été découpée en 7 partitions :

| Adresse | Nom | Taille | Permission |
|---|---|---|---|
| 0x000000000000-0x000000040000 | u-boot | 256K | Read Only[2] |
| 0x000000040000-0x000000044000 | u-boot_env | 16K | Read Only[2] |
| 0x000000044000-0x000000244000 | recovery_kernel | 2M | Read Only[2] |
| 0x000000244000-0x000000444000 | kernel | 2M | Writable |
| 0x000000444000-0x000000844000 | recovery_fs | 4M | Read Only[2] |
| 0x000000844000-0x000003a44000 | rootfs | 50M | Writable |
| 0x000003a44000-0x000004000000 | userspace | ~5.5M[1] | Writable |

[1] : La taille finale de l'espace utilisateur dépend des mauvais blocs de la mémoire NAND, mais il sera toujours au moins 5M.
[2] : en lecture seule (à partir de la carte) par mesure de sécurité

L'espace utilisateur est limité à une taille d'environ 5Mo.

## *Procédure de démarrage*

Le processus de démarrage à la mise sous tension est le suivant :

- Le processeur charge les 4 premiers octets de la mémoire NAND dans Steppingstone et l'exécute
- SteppingStone copie u-boot de la mémoire NAND dans la RAM et lance U-Boot
- U-Boot initialise le processeur, définit l'état de GPIO, etc, copie le noyau Linux en RAM, initialise les arguments pour le noyau, vérifie que l'image du noyau est valide et la démarre
- Linux démarre, initialise l'USB, le réseau, la mémoire NAND, etc, puis monte la partition rootfs de la mémoire NAND sur /
- init se lance comme le père de tous les processus, lit le fichier /etc/inittab pour mettre en place le système. Cela comprend le montage des autres systèmes de fichiers, les paramètres d'environnement de l'hôte, et l'exécution des scripts dans /etc/init.d, entre autres
- inittab exécute ensuite toutes les applications personnalisées
- inittab établit alors un getty sur le premier port série, prêt pour être utilisé par l'interface de débogage

*Remarque :*

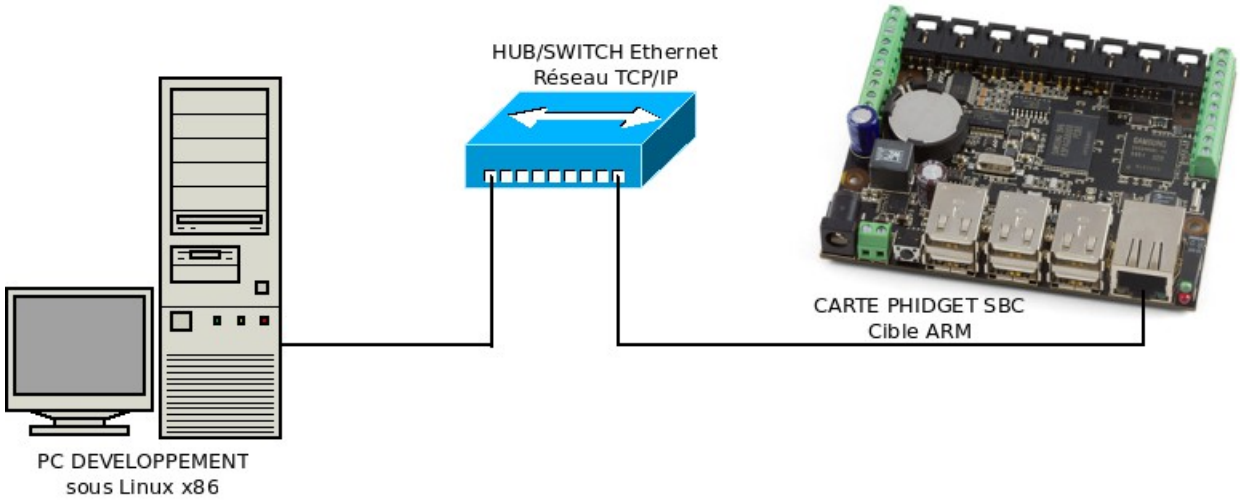
*Un chargeur d'amorçage (ou bootloader) est un logiciel permettant de lancer un (ou plusieurs) système d'exploitation. U-Boot (the Universal Boot Loader) est un bootloader pour système embarqué.*

*Lien : http://www.denx.de/wiki/U-Boot/WebHome*

# L'environnement de développement

## Mise en situation

On propose de mettre en oeuvre l'environnement de développement suivant :




Le développeur utilisera un PC x86 sous Linux comme poste de travail : c'est le poste de développement. Il devra développer sur ce poste des programmes destinés à un système embarquée : ici la carte PhidgetSBC équipée d'un processeur ARM. Il est donc nécessaire de mettre en place une chaîne de développement croisée permettant de développer du code pour une cible ARM.

## Compilation croisée

Un compilateur croisé (cross compiler) est un programme capable de traduire un code source en code objet ayant un environnement d'exécution (architecture matérielle / système d'exploitation) différent de celui où la compilation est effectuée. Ces compilateurs sont principalement utilisés en informatique industrielle.

## Chaîne de compilation traditionnelle sous Linux

La chaîne de compilation GNU utilise les composants suivants:

- Le compilateur qui constitue le paquetage gcc.
- Les outils annexes (assembleur, éditeur de liens, etc.) qui constituent le paquetage binutils.
- La GNULibc qui constitue le paquetage glibc. Ce paquetage contient en général la bibliothèque de gestion des threads).

Dans le cas d'un système Linux x86, ces paquetages sont installés sur le système sous forme de paquetages RPMS (Redhat, Mandriva, ...) ou DEB (Debian, Ubuntu, ...).

Dans le cas de la compilation croisée, ces différents outils, ainsi que le compilateur, seront exécutés dans un environnement Linux x86 (le poste de développement) mais le code généré sera d'un type différent, ici pour un processeur ARM.

La génération de la chaîne de compilation croisée est réalisable "à la main" mais elle est souvent fastidieuse car elle peut nécessiter l'application de « patch » sur un ou plusieurs paquetage en fonction des différentes architectures. De ce fait, on utilisera des outils spécialisés "prêts à l'emploi".

## *Buildroot*

La carte PhidgetSBC est un système embarqué qui exécute le système d'exploitation Linux. La distribution Linux embarquée a été construite en utilisant Buildroot. Elle fournit un accès shell complet via un serveur SSH, un accès de configuration par une interface web via un serveur HTTP, des outils de développement (gcc), le débogueur gdb et des outils de ligne de commande standard (busybox).

Buildroot est nécessaire pour :

- le développement d'application embarquée en C/C++
- la construction d'un nouveau noyau personnalisé

Le Buildroot utilisé par la carte PhidgetSBC peut être récupéré sur le site web de la société :

- http://www.phidgets.com/drivers.php
- http://www.phidgets.com/old_drivers.php (pour les anciennes versions)

Il peut s'avérer important d'identifier la version utilisée sur la carte phidget préalablement :

```
# cat /etc/phidgetsbc_version
version 1.0.3
type full
date 20100304
```

*Remarque : ceci correspond au firmware installé et livré en standard pour l'achat d'une carte PhidgetSBC 1070 début 2010.*

Puis, télécharger le buildroot correspondant :

```
$ wget http://www.phidgets.com/downloads/libraries/buildroot-
phidgetsbc_1.0.3.20100304.tar.gz
```

*Remarque : Buildroot est un outil qui permet l'automatisation du processus de construction d'une chaîne de compilation croisée et d'un système Linux embarqué complet. Il automatise le processus de téléchargement, configuration, compilation et installation de tous les composants d'un système Linux embarqué : chaîne de compilation croisée, chargeur de démarrage, noyau, système de fichiers racine avec BusyBox, bibliothèques graphiques, réseau, multimédia, etc.*

- Lien : http://www.buildroot.org/ et http://buildroot.uclibc.org/downloads/snapshots/
- Lien : http://www.arm.linux.org.uk/docs/

L'installation est décrite dans la partie A . 2.

# Manipulations basiques

## Partie A : manuel d'installation

### A . 1 . Installer le matériel fourni

On réalise dans l'ordre les actions suivantes :

- relier la carte phidget 107x au HUB/Switch par un câble Ethernet
- alimenter la phidget 107x (adaptateur xxV fourni)
- relier le poste de développement PC_DEV au HUB/Switch par un câble Ethernet
- démarrer le poste de développement PC_DEV sous Linux

### A . 2 . Installer la chaîne de développement croisé

Il faut préalablement s'identifier sous le compte root. Votre poste doit être configuré pour permettre la fabrication du buildroot. Pour cela, il faut vous consulter la fiche de pré-installation en Annexe 4. Puis :

```
# cd /usr/local/
# wget http://www.phidgets.com/downloads/libraries/buildroot-
phidgetsbc_1.0.4.20110204.tar.gz
# tar zxvf buildroot-phidgetsbc_1.0.4.20110204.tar.gz
# ln -s buildroot-phidgetsbc_1.0.4.20110204 buildroot
# ls -l
# cd buildroot
# ls -al
```

Le noyau a été pré-configuré par la société Phidget pour sa carte 107x. Il est tout de même possible de le personnaliser :

```
# make menuconfig
```

*Remarque : si vous envisagez de produire des applications en C++ pour la carte PhidgetSBC, il vous faudra activer son support à partir de ce menu. Quelques captures d'écran sont disponibles en annexe 6.*

Les options de configuration sont dans le fichier .config :

```
# vim .config
```

*Remarque : pour connaître les principales options, consulter l'annexe 7.*

Pour fabriquer un nouveau noyau (firmware), télécharger les outils de développements nécessaires et fabriquer la chaîne de compilation croisée, il faut faire :

```
# make
```

*Remarque : attention cette étape peut d'avérer assez longue !*

Une fois la fabrication terminée avec succès :

- Les outils de développement de la chaîne de compilation croisée sont maintenant disponibles ici :

```
# ls -l /usr/local/buildroot/output/staging/usr/bin/
```

- Le noyau (firmware) .bin se trouve dans ce répertoire :

```
# ls -l /usr/local/buildroot/output/images/
```

- La racine du système de fichiers qui sera installé sur la carte phidget :

```
# ls -l /usr/local/buildroot/output/target/
```

Pour l'instant, nous avons juste besoin de la chaîne de compilation croisée pour développer nos propres applications pour la carte PhidgetSBC. Il est tout à fait possible d'installer le nouveau noyau (flash du firmware) sur la carte. Pour cela, il vous faut consulter la documentation fournie par Phidget.

## *Partie B : mise en oeuvre de la communication réseau*

*Attention : les adresses réseaux utilisées dans ce document sont fournies à titre d'exemple. Ces adresses peuvent être différentes suivant votre configuration.*

### B . 1 . Identifier la configuration réseau du poste de développement.

Pour obtenir la configuration réseau du poste, taper la commande :

```
# ifconfig
```

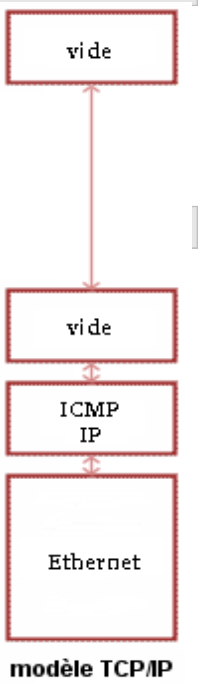
### B . 2 . Si nécessaire, configurer les adresses IP des postes à mette en oeuvre.

*Remarque : la configuration la plus courante est de placer les deux équipements sur le même réseau IP (liaison directe ou par l'intermédiaire d'un hub/switch). Pour configurer la carte, il vous faut consulter la documentation fournie par Phidget.*

### B . 3 . Tester la liaison réseau avec le système embarqué (carte Phidget).

Pour réaliser un test de base d'une liaison réseau, on utilise souvent la commande ping qui permettra, en cas de succès, de valider la pile de protocoles jusqu'au niveau IP.

*Remarques : **ping** est le nom d'une commande informatique réseau (développée par Mike Muuss) permettant d'envoyer une requête ICMP (demande d'ECHO ou echo-request) d'une machine à une autre machine et attend en retour une réponse d'ECHO (echo-reply). Selon la réponse on connaît l'état de la machine distante. Cette commande réseau de base permet d'obtenir des informations et en particulier le temps de réponse de la machine à travers le réseau et aussi quel est l'état de la connexion avec cette machine (renvoi d'un code d'erreur correspondant).*



modèle TCP/IP

***ICMP*** *(Internet Control Message Protocol - Protocole de message de contrôle sur Internet ) est un protocole de niveau 3 sur le modèle OSI, qui permet le contrôle des erreurs de transmission. En effet, comme le protocole IP ne gère que le transport des paquets et ne permet pas l'envoi de messages d'erreur, c'est grâce à ce protocole qu'une machine émettrice peut savoir qu'il y a eu un incident de réseau (par exemple lorsqu'un service ou un hôte est inaccessible). Il est détaillé dans la RFC 792.*

En utilisant l'interpréteur de commandes, taper la commande :

```
ping adresse_ip_phidget
```

### B . 4 . Tester la communication réseau avec le système embarqué.

La carte embarquée PhidgetSBC ne disposant pas de terminal opérateur (clavier/écran), on va donc utiliser celui du poste de développement par l'intermédiaire d'un logiciel d'émulation de terminal via le réseau. Le protocole permettant l'émulation de terminal en réseau se nomme ssh. Le logiciel Putty ou tout simplement la commande ssh va nous permettre d'ouvrir une session (accès à un terminal de connexion via le réseau sur la carte phidget).

#### B . 4 . 1 . Ouvrir une session sur la carte phidget.

L'ouverture d'une session sur la carte phidget nécessite une authentification (un compte identifié par un login et un mot de passe). On utilisera les paramètres de connexion suivants :

```
login : root ou user
password : password

Host Name (or IP address) ← indiquer l'adresse IP de la carte
Port ← 22 (le numéro de port par défaut du service ssh)

$ ssh [-p port] [utilisateur@]hostname
```

Une fois connecté, il est possible d'exécuter des commandes sur la carte phidget. Ceci est possible car celle-ci dispose d'un système d'exploitation (Linux).

#### B . 4 . 2 . Lister les fichiers et répertoires de la carte

Taper les commandes :

```
# ls

# pwd
```

Pour mettre fin à une session ssh, il suffit de taper la commande :

```
# exit
```

## *Partie C : fabrication d'un exécutable pour la carte TINI*

### C . 1 . Configurer l'environnement pour le développement

Pour compiler une application destinée à la carte PhidgetSBC, il vous faut pouvoir accéder aux outils de développement spécifiques à une architecture ARM. Ces outils sont situés ici :

```
# ls -l /usr/local/buildroot/output/staging/usr/bin/
```

On va tout simplement mettre à jour la variable d'environnement PATH :

```
export BUILDROOT_PATH=/usr/local/buildroot
export PATH=$BUILDROOT_PATH/output/staging/usr/bin/:$PATH
```

Vérifier la présence des deux compilateurs :

```
# arm-linux-gcc
arm-linux-gcc: no input files

# arm-linux-g++
arm-linux-g++: no input files
```

### C . 2 . Éditer les fichiers sources de test

```
# vim hello.c
#include <stdio.h>

int main()
{
   printf("Hello world !\n");
   return 0;
}
```

```
# vim hello.cpp
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello world !" << endl;
   return 0;
}
```

### C . 3 . Fabriquer les exécutables

On va maintenant éditer un fichier Makefile pour la compilation en langage C et lancer sa fabrication avec make :

```
# vim Makefile
CC=arm-linux-gcc
NM=arm-linux-nm
AR=arm-linux-ar
CXX=arm-linux-g++
CPP=arm-linux-cpp
LD=arm-linux-ld
STRIP=arm-linux-strip

#EXTRALIBS=-lphidget21
#DEBUGFLAGS := -g -O0 -D_DEBUG -DDEBUG

LIBS := -lm -lpthread -ldl $(EXTRALIBS)
CFLAGS := -Wall -Wno-strict-aliasing $(DEBUGFLAGS) -fpic -D_LINUX -I.

hello_c: hello.c
   $(CC) $(CFLAGS) $(LIBS) -o hello_c hello.c
   $(STRIP) hello_c

all: hello_c

clean:
      rm *~ *.o hello_c

# make
arm-linux-gcc -Wall -Wno-strict-aliasing  -fpic -D_LINUX -I. -lm -lpthread -ldl
-o hello_c hello.c
arm-linux-strip hello_c
```

*Remarque : l'utilitaire **strip** permet de réduire la taille d'un fichier exécutable ce qui est conseillé dans le cas d'un système embarqué où les ressources sont toujours limitées.*

Et un fichier Makefile pour la compilation en langage C++ et lancer sa fabrication avec make :

```
# vim Makefile
CC=arm-linux-gcc
NM=arm-linux-nm
AR=arm-linux-ar
CXX=arm-linux-g++
CPP=arm-linux-cpp
LD=arm-linux-ld
STRIP=arm-linux-strip

#EXTRALIBS=-lphidget21
#DEBUGFLAGS := -g -O0 -D_DEBUG -DDEBUG

LIBS := -lm -lpthread -ldl $(EXTRALIBS)
CFLAGS := -Wall -Wno-strict-aliasing $(DEBUGFLAGS) -fpic -D_LINUX -I.

all: hello_cpp hello_cpp_s

hello_cpp: hello.cpp
        $(CXX) $(CFLAGS) $(LIBS) -o hello_cpp hello.cpp
```

```
        $(STRIP) hello_cpp

hello_cpp_s: hello.cpp
        $(CXX) -static -o hello_cpp_s hello.cpp
        $(STRIP) hello_cpp_s

clean:
           rm *~ *.o hello_cpp hello_cpp_s

# make
arm-linux-g++ -Wall -Wno-strict-aliasing  -fpic -D_LINUX -I. -lm -lpthread -ldl
-o hello_cpp hello.cpp
arm-linux-strip hello_cpp
arm-linux-g++ -static -o hello_cpp_s hello.cpp
arm-linux-strip hello_cpp_s
```

*Remarque : à titre d'exemple, le Makefile ci-dessus permet les deux éditions de liens disponibles : dynamique (par défaut) et statique.*

Un fichier Makefile « générique » pour le C++ :

```
# vim Makefile
CC=arm-linux-gcc
NM=arm-linux-nm
AR=arm-linux-ar
CXX=arm-linux-g++
CPP=arm-linux-cpp
LD=arm-linux-ld
STRIP=arm-linux-strip

#EXTRALIBS=-lphidget21
#DEBUGFLAGS := -g -O0 -D_DEBUG -DDEBUG

LIBS := -lm -lpthread -ldl $(EXTRALIBS)
CFLAGS := -Wall -Wno-strict-aliasing $(DEBUGFLAGS) -fpic -D_LINUX -I.
LDFLAGS = $(LIBS)

.PHONY: clean copy

SRCFILES := $(wildcard *.cpp)
OBJFILES := $(patsubst %.cpp, %.o, $(SRCFILES))

TARGET = le nom de l'application à construire

all: $(TARGET)

%.o: %.cpp
   $(CXX) $< $(CFLAGS) -o $@

$(TARGET): $(OBJFILES)
   $(CXX) $^ $(LDFLAGS) -o $@
   $(STRIP) $(TARGET)

clean:
   rm $(TARGET) $(OBJFILES) *~

# make
```

### C . 4 . Vérifier les exécutables fabriqués.

Pour obtenir des informations sur les exécutables, il suffit d'utiliser la commande file :

```
# file hello_c
../hello_c/hello_c: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically
linked (uses shared libs), stripped

# file hello_cpp
hello_cpp: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked
(uses shared libs), stripped

# file hello_cpp_s
hello_cpp_s: ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically linked,
stripped
```

Le code produit est bien au format ELF 32 bits pour une architecture ARM. Le fichier produit est bien un exécutable ARM. Il suffit maintenant de le transférer sur la cible puis l'exécuter.

*Remarques :*

*ELF (Executable and Linking Format) est un format de fichier informatique binaire utilisé pour l'enregistrement de code compilé (objets, exécutables, bibliothèques de fonctions). La commande man elf donnera de plus amples informations. Aujourd'hui, ce format est utilisé dans la plupart des systèmes d'exploitation Unix (GNU/Linux, Solaris, Irix, System V, BSD), à l'exception de Mac OS X qui utilise le format Mach-O. Windows utilise le format PE (Portable Executable).*

*Dans le cas des exécutables liés dynamiquement, il est aussi possible de voir les dépendances vis à vis des librairies en utilisant la commande arm-linux-ldd.*

## *Partie D : déploiement et exécution d'une application*

### D . 1 . Déploiement de l'application sur le système embarqué cible.

On utilisera SCP pour déployer une application sur le carte phidget :

```
$ scp [-P port] [[user@]host1:]file1 [[user@]host2:]file2

password:

file                                    100%   22KB  21.9KB/s   00:00
```

Il faut déployer les trois exécutables : hello_c, hello_cpp et hello_cpp_s.

### D . 2 . Exécution d'une application.

Il faut ouvrir une session ssh et lancer l'exécution du programme transféré préalablement.

Taper les commandes :

```
$ ssh [-p port] [utilisateur@]hostname

password:
```

```
# ./hello_c
Hello world !

# ./hello_cpp
Hello world !

# ./hello_cpp_s
Hello world !
```

On peut vérifier les dépendances vis à vis des librairies dynamiques pour les trois exécutables :

```
# ldd ./hello_c
        libm.so.0 => /lib/libm.so.0 (0x4000e000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x40026000)
        libdl.so.0 => /lib/libdl.so.0 (0x40040000)
        libc.so.0 => /lib/libc.so.0 (0x4004b000)
        ld-uClibc.so.0 => /lib/ld-uClibc.so.0 (0x40000000)

# ldd ./hello_cpp
        libpthread.so.0 => /lib/libpthread.so.0 (0x4000e000)
        libdl.so.0 => /lib/libdl.so.0 (0x40028000)
        libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0x40033000)
        libm.so.0 => /lib/libm.so.0 (0x400a8000)
        libgcc_s.so.1 => /lib/libgcc_s.so.1 (0x400c0000)
        libc.so.0 => /lib/libc.so.0 (0x400d3000)
        ld-uClibc.so.0 => /lib/ld-uClibc.so.0 (0x40000000)

# ldd ./hello_cpp_s
        not a dynamic executable
```

*Remarque : l'exécutable hello_cpp compilé en C++ et lié dynamiquement est dépendant des deux librairies libstdc++.so.6 et libgcc_s.so.1. Il est possible que ces deux bibliothèques ne soient pas présentes sur la carte Phidget. Pour cela il faut appliquer la procédure décrite ci-dessous.*

### D . 3 . Installation des librairies dynamiques pour le développement C++

Si les deux librairies sont manquantes sur votre carte, vous obtiendrez ce type de message :

```
# ldd ./hello_cpp
        libpthread.so.0 => /lib/libpthread.so.0 (0x4000e000)
        libdl.so.0 => /lib/libdl.so.0 (0x40028000)
        libstdc++.so.6 => not found
        libm.so.0 => /lib/libm.so.0 (0x40033000)
        libgcc_s.so.1 => not found
        libc.so.0 => /lib/libc.so.0 (0x4004b000)
        ld-uClibc.so.0 => /lib/ld-uClibc.so.0 (0x40000000)
```

*Remarque : le message « not found » indique les librairies ne sont « trouvables » sur la carte. Cela ne sert bien évidemment à rien de lancer cet exécutable.*

On va installer ces deux librairies manquantes sans « reflasher » un nouveau firmware. Pour cela, il faut préalablement remonter le système de fichiers racine de la carte phidget en lecture/écriture :

```
# mount -o remount,rw /
```

Puis, on va copier les deux librairies vers la carte en utilisant scp à partir du poste de développement (indiquer l'adresse ip de la carte phidget en remplacement de host) :

```
# scp /usr/local/buildroot/output/target/lib/libgcc_s.so.1
root@host:/lib/libgcc_s.so.1

# scp /usr/local/buildroot/output/target/usr/lib/libstdc++.so.6.0.10
root@host:/usr/lib/libstdc++.so.6.0.10
```

Ensuite, on ouvre une session ssh et on réalise les actions suivantes :

```
# cd /usr/lib
# chmod 755 libstdc++.so.6.0.10
# ln -s libstdc\+\+.so.6.0.10 libstdc\+\+.so.6
# ln -s libstdc\+\+.so.6.0.10 libstdc\+\+.so

# cd /lib
# chmod 755 libgcc_s.so.1
# ln -s libgcc_s.so.1 libgcc_s.so
# ldconfig

# ldconfig -v
```

On peut vérifier maintenant que les dépendances sont bien résolues :

```
# ldd ./hello_cpp
        libpthread.so.0 => /lib/libpthread.so.0 (0x4000e000)
        libdl.so.0 => /lib/libdl.so.0 (0x40028000)
        libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0x40033000)
        libm.so.0 => /lib/libm.so.0 (0x400a8000)
        libgcc_s.so.1 => /lib/libgcc_s.so.1 (0x400c0000)
        libc.so.0 => /lib/libc.so.0 (0x400d3000)
        ld-uClibc.so.0 => /lib/ld-uClibc.so.0 (0x40000000)
```

# Annexe 1 : Obtenir des informations sur le système embarqué

Une liste d'informations utiles pour établir la fiche de configuration de la carte PhidgetSBC :

| *Quoi ?* | *Comment ?* |
|---|---|
| Nom de la machine | **# echo $HOSTNAME**<br>**# hostname** |
| Version de la distribution actuellement installée | **# cat /etc/phidgetsbc_version** |
| Version du noyau GNU/Linux actuellement installé | **# cat /proc/version**<br>**# uname -a**<br>**# uname -r** |
| Information sur le processeur | **# cat /proc/cpuinfo** |
| Informations sur la mémoire (MemTotal, SwapTotal, ...) | **# cat /proc/meminfo**<br>**# free** |
| Partitions et montages | **# cat /proc/mounts**<br>**# cat /proc/partitions**<br>**# cat /etc/fstab**<br>**# cat /etc/mtab**<br>**# mount** |
| Espace disque des systèmes de fichiers | **# df**<br>**# du -sh /** |
| Les modules | **# cat /proc/modules**<br>**# lsmod puis modinfo** |
| Le bus USB | **# lsusb**<br>**# cat /proc/bus/usb/devices**<br>**# cat /proc/bus/usb/ ... etc ...** |
| Réseau | **# ifconfig**<br>**# route**<br>**# netstat**<br>**# cat /etc/config/network.conf**<br>**# cat /etc/config/wireless_network.conf** |
| Librairies installées | **# ldconfig -v**<br>**# ls /lib et /usr/lib**<br>**# ldd /usr/lib/*.so**<br>**# nm -s /usr/lib/*.so et /usr/lib/*.a** |
| Démarrage du système | **# dmesg**<br>**# cat inittab**<br>**# ls init.d/** |
| Environnement | **# cat /etc/profile**<br>**# env**<br>**# set** |
| Liste des commandes et version de la BusyBox | **# busybox**<br>**# ls /bin**<br>**# ls /usr/bin** |
| Spécifique Phidget | **# phidget21version**<br>**# phidgetlist** |

## *Exemple de fiche de configuration de la carte*

| *Fiche d'identité : PhidgetSBC 1070* | |
|---|---|
| **Remplie par : _____** | **Date :** |
| Numéro de la carte (id) | |
| Nom de la machine (hostname) | |
| Distribution (et version) actuellement installée | |
| Version du noyau GNU/Linux actuellement installé | |
| Information sur le processeur | |
| Informations sur la mémoire | |
| Partitions et montages | |
| Disque(s) dur(s) | |
| Espace disque des systèmes de fichiers | |
| Le bus USB | |
| Réseau | |
| Version de la BusyBox | |
| Spécifique Phidget | |
| **Installation à réaliser : ☐ Linux ☐ WindowsCE ☐ Aucune ☐ Autre :** | |
| Distribution (et version) à installer | |

# Annexe 2 : Glossaire logiciel

Quelques outils logiciels mis en oeuvre :

| *Nom* | *Description* | *Rôle* |
|---|---|---|
| Buildroot | Buildroot est un outil qui permet l'automatisation du processus de construction d'une chaîne de compilation croisée et d'un système Linux embarqué complet. Il automatise le processus de téléchargement, configuration, compilation et installation de tous les composants d'un système Linux embarqué : chaîne de compilation croisée, chargeur de démarrage, noyau, système de fichiers racine avec BusyBox, bibliothèques graphiques, réseau, multimédia, etc.<br>Lien : http://www.buildroot.org/ | permet la construction d'un système Linux embarqué et fournit la chaîne de compilation croisée |
| uClibc | uClibc est une petite bibliothèque standard de C destinée au développement de systèmes embarqués avec Linux. UClibc est en particulier plus petite que la glibc généralement utilisée sur Linux. uClibc est une bibliothèque libre distribuée selon les termes de la GNU LGPL.<br>Elle supporte les plateformes suivantes : alpha, amd64, ARM, Blackfin, cris, h8300, hppa, i386, i960, ia64, m68k, mips/mipsel, PowerPC, SH, SPARC, et v850. uClibc supporte également les systèmes sans MMU.<br>Lien : http://www.uclibc.org/ | offre le support de la bibliothèque standard de C pour le développement de programmes destinés à un système embarqué |
| BusyBox | BusyBox est un logiciel qui implémente un grand nombre des commandes standards sous Unix, à l'instar des GNU Core Utilities. BusyBox est conçu comme un unique fichier exécutable, ce qui le rend très adapté aux distributions Linux utilisées sur les systèmes embarqués. Comme chaque fichier binaire exécutable pour Linux comporte plusieurs kilo-octets d'informations additionnelles, l'idée de combiner plus de deux cents programmes en un seul fichier exécutable permet de gagner une taille considérable. Distribué sous la licence GNU GPL version 2, BusyBox est un logiciel libre.<br>Lien : http://www.busybox.net/ | permet l'accès aux commandes usuels d'un système Linux |
| U-Boot | Un chargeur d'amorçage (ou bootloader) est un logiciel permettant de lancer un (ou plusieurs) système d'exploitation. U-Boot (the Universal Boot Loader) est un bootloader pour système embarqué.<br>Lien : http://www.denx.de/wiki/U-Boot/WebHome | permet le chargement et le démarrage du système d'exploitation Linux sur la carte embarquée |
| SSH | Secure Shell (SSH) est à la fois un programme informatique et un protocole de communication sécurisé. Le protocole SSH a été conçu avec l'objectif de remplacer les différents programmes rlogin, telnet et rsh. Habituellement le protocole SSH utilise le port TCP 22. Il est particulièrement utilisé pour ouvrir un shell (console) sur un ordinateur distant. | permet d'ouvrir, à partir d'un PC, une session SSH sur la carte phidgetsbc et donc d'interagir avec celle-ci. |
| SCP | Secure copy (SCP) désigne un transfert sécurisé de fichiers entre deux ordinateurs utilisant le protocole de communication SSH. Le terme SCP désigne aussi bien le programme scp que le protocole scp. Le protocole SCP implémente uniquement le transfert de fichiers. Pour cela, une connexion est établie en utilisant SSH, puis un serveur SCP est lancé. Le programme serveur est le même que le programme client. Lors d'un téléchargement montant (download), le client envoie au serveur les fichiers à télécharger en comprenant éventuellement leurs attributs de base (permissions, horodatage). C'est un avantage par rapport au protocole FTP, qui ne prévoit pas d'inclure les attributs lors d'un téléchargement. | permet de transférer des fichiers du poste de développement (PC_DEV) vers la carte embarquée (phidgetsbc). |

# Annexe 3 : Exemple de tests de validation

Ces tests permettent de valider la mise en oeuvre du matériel, des logiciels et des ressources de développement constituant le système.

| *Mise en oeuvre : tests de validation* | *OUI/NON* |
|---|---|
| La liaison réseau IP PC-PHIDGET est-elle fonctionnelle ? | |
| La communication réseau PC-PHIDGET par ssh est-elle fonctionnelle ? | |
| La fabrication d'un exécutable ARM est-elle fonctionnelle ? | |
| Le transfert de fichiers PC-PHIDGET par scp est-elle fonctionnelle ? | |
| L'exécution d'une application ARM sur la cible est-elle fonctionnelle ? | |
| Fait le ___ / ___ / _____ par _____<br><br>*Remarques :* | |

# Annexe 4 : Pré-installation sur le poste de développement

Après avoir « désarchiver le buildroot » avec tar, il vous faut lire le fichier README. Celui précise notamment les outils dont vous devez disposer sur votre poste de développement afin de fabriquer le buildroot, celui fournit par la société Phidget étant minimum.

```
# pwd
/usr/local/buildroot-phidgetsbc_1.0.4.20110204

# vim README_PHIDGETSBC
```

*Remarque : la distribution utilisée par la société Phidget semble être basée sur une Debian mais cela fonctionne aussi sur une Mandriva.*

```
Extrait du fichier README_PHIDGETSBC :
The following packages are required to be installed to build the 'full' Buildroot
system (install with 'sudo apt-get install'):
libncurses5-dev, bison, flex, gettext, subversion, uboot-mkimage, acl-dev
libacl1-dev, zlib1g-dev, kaffe -or- sun-java6-jdk -or- openjdk-6-jdk,
antlr, liblzo2-dev, texinfo
```

Avec une mandriva, il vous faudra utiliser l'outil **urpmi** sous le compte root pour installer les packages manquants.

*Remarque : Il faudra aussi ajouter la glibc-static-devel si la fabrication du buildroot est aussi configurée pour une compilation statique.*

Pour vérifier leur présence sous Mandriva, vous pouvez réaliser les commandes suivantes :

```
# rpm -qa | grep -i libncurses-devel
# rpm -qa | grep -i bison
# rpm -qa | grep -i flex
# rpm -qa | grep -i gettext
# rpm -qa | grep -i subversion
# rpm -qa | grep -i libacl-devel
# rpm -qa | grep -i zlib1-devel
# rpm -qa | grep -i kaffe
# rpm -qa | grep -i liblzo-devel
# rpm -qa | grep -i texinfo
# rpm -qa | grep -i glibc-static-devel
```

Le paquetage uboot-mkimage n'étant pas disponible dans les dépôts Mandriva, il vous faudra le télécharger puis l'installer comme ceci :

```
# wget ftp://mirror.switch.ch/pool/1/mirror/epel/beta/6/i386/uboot-tools-2010.03-
1.el6.i686.rpm

# rpm -ivh uboot-tools-2010.03-1.el6.i686.rpm
```

Attention si vous installez la version buildroot-phidgetsbc_1.0.4.20110204.tar.gz, il vous faudra peut-être modifier les numéros de version de ces packages :

```
# vim package/libphidget/libphidget.mk
LIBPHIDGET_VERSION:=2.1.7.20110203

# vim package/phidgetwebservice/phidgetwebservice.mk
PHIDGETWEBSERVICE_VERSION:=2.1.7.20110203
```

Le buildroot fournit par la société Phidget étant minimum (8,5 Mio), un certain nombre d'outils seront téléchargés pendant la fabrication par make. Il vous faut évidemment une connexion Internet pendant cette opération.

Les archives téléchargées pendant la fabrication sont :

```
# ls /usr/local/buildroot/dl/
autoconf-2.65.tar.bz2              linux-2.6.32.14.tar.bz2
automake-1.10.tar.bz2              lockfile-progs_0.1.11-0.1.tar.gz
binutils-2.20.1.tar.bz2            lzo-2.03.tar.gz
busybox-1.16.1.tar.bz2             m4-1.4.9.tar.bz2
dropbear-0.52.tar.gz               mDNSResponder-107.6.tar.gz
e2fsprogs-1.41.11.tar.gz           module-init-tools-3.2.2.tar.bz2
fakeroot_1.9.5.tar.gz              mpfr-2.4.1.patch
gcc-4.3.4.tar.bz2                  mpfr-2.4.1.tar.bz2
genext2fs-1.4.tar.gz               mtd-utils-1.3.1.tar.bz2
gmp-4.2.4.tar.bz2                  openntpd-3.9p1.tar.gz
haserl-0.9.26.tar.gz               phidgetwebservice_2.1.7.20110203.tar.gz
ifplugd-0.28.tar.gz                pkg-config-0.23.tar.gz
jpegsrc.v6b.tar.gz                 u-boot-2010.03.tar.bz2
libdaemon-0.13.tar.gz              uClibc-0.9.31.tar.bz2
liblockfile_1.08.orig.tar.gz       udev-114.tar.bz2
libphidget_2.1.7.20110203.tar.gz   usbmount_0.0.14.1.tar.gz
libtool-1.5.24.tar.gz              wireless_tools.29.tar.gz
libusb-1.0.3.tar.bz2               wpa_supplicant-0.6.10.tar.gz
libusb-compat-0.1.3.tar.bz2        zlib-1.2.3.tar.bz2
```

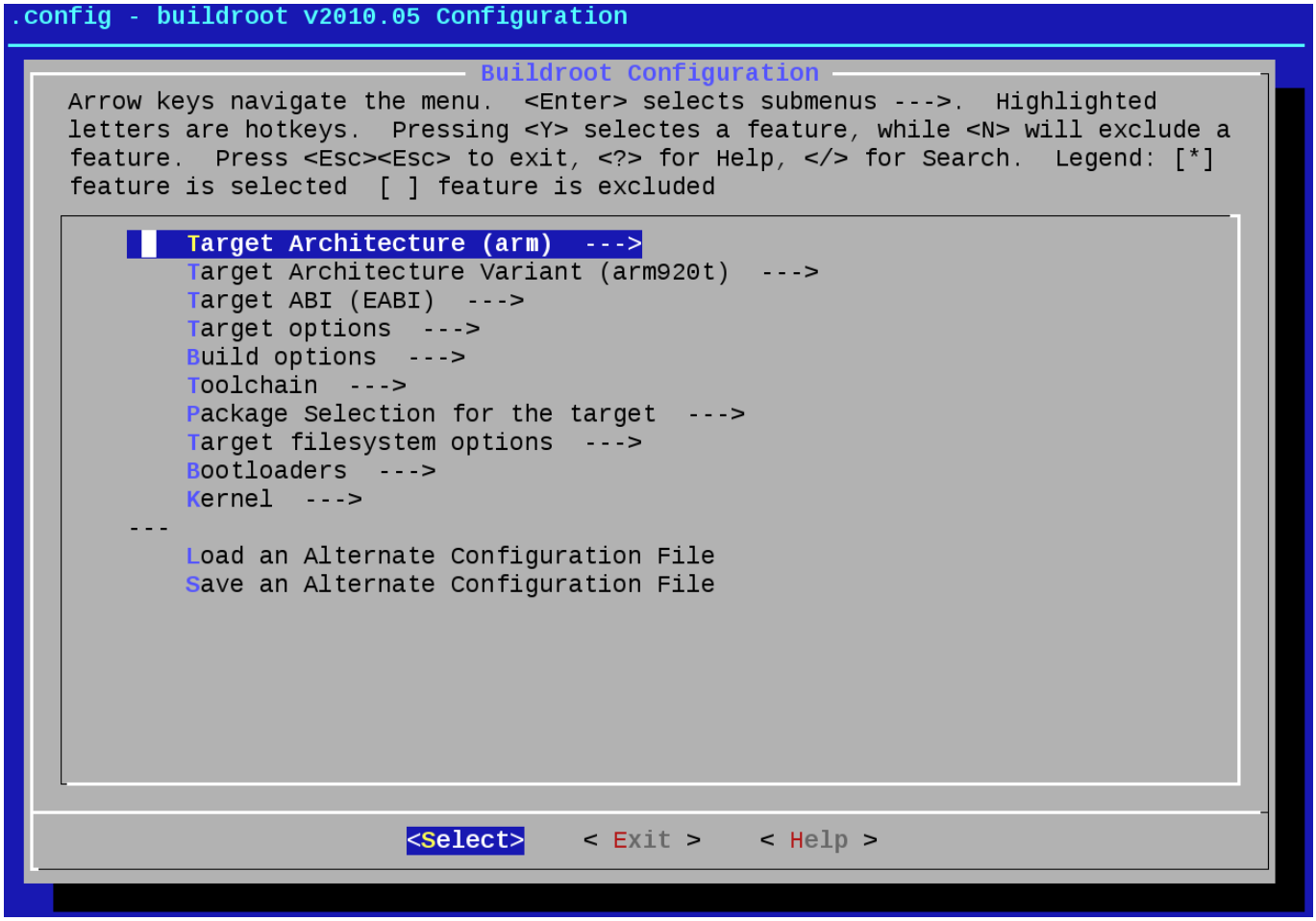
# Annexe 5 : Données persistantes, Journalisation (log)

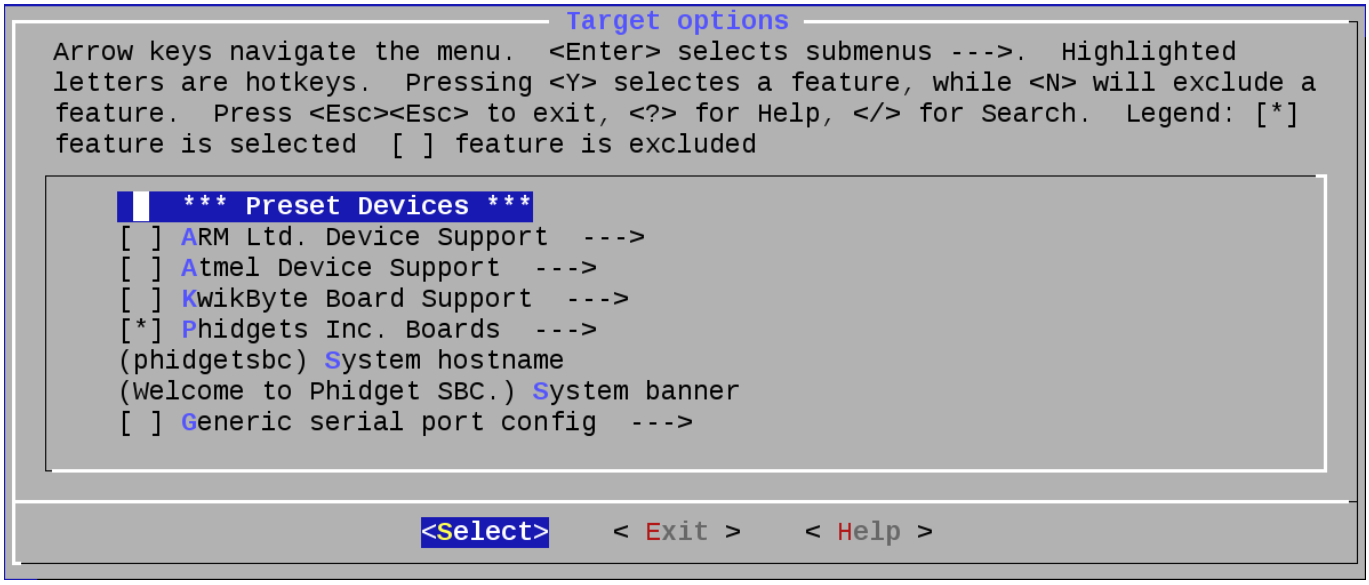
Si vous avez besoin d'enregistrer des données à partir d'une application personnelle, vous pouvez soit :

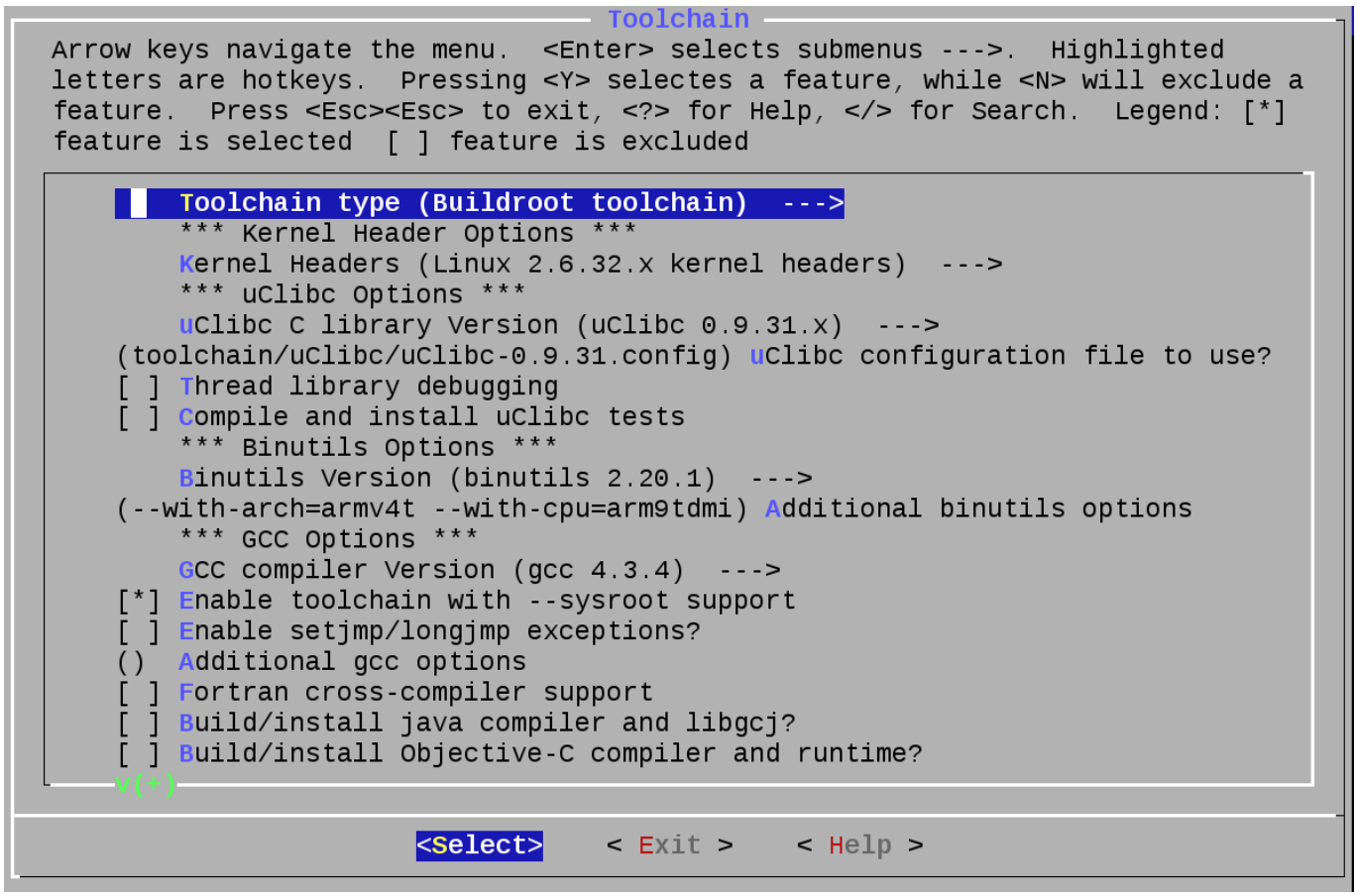
- les enregistrer directement dans le répertoire de l'application mais attention avec la limite de taille de l'espace utilisateur (environ 5Mo)
- les enregistrer dans /tmp mais les données seront effacées au redémarrage
- utiliser une clé USB ou un média de type mémoire flash (montage automatiquement dans /media/usb[0-9])

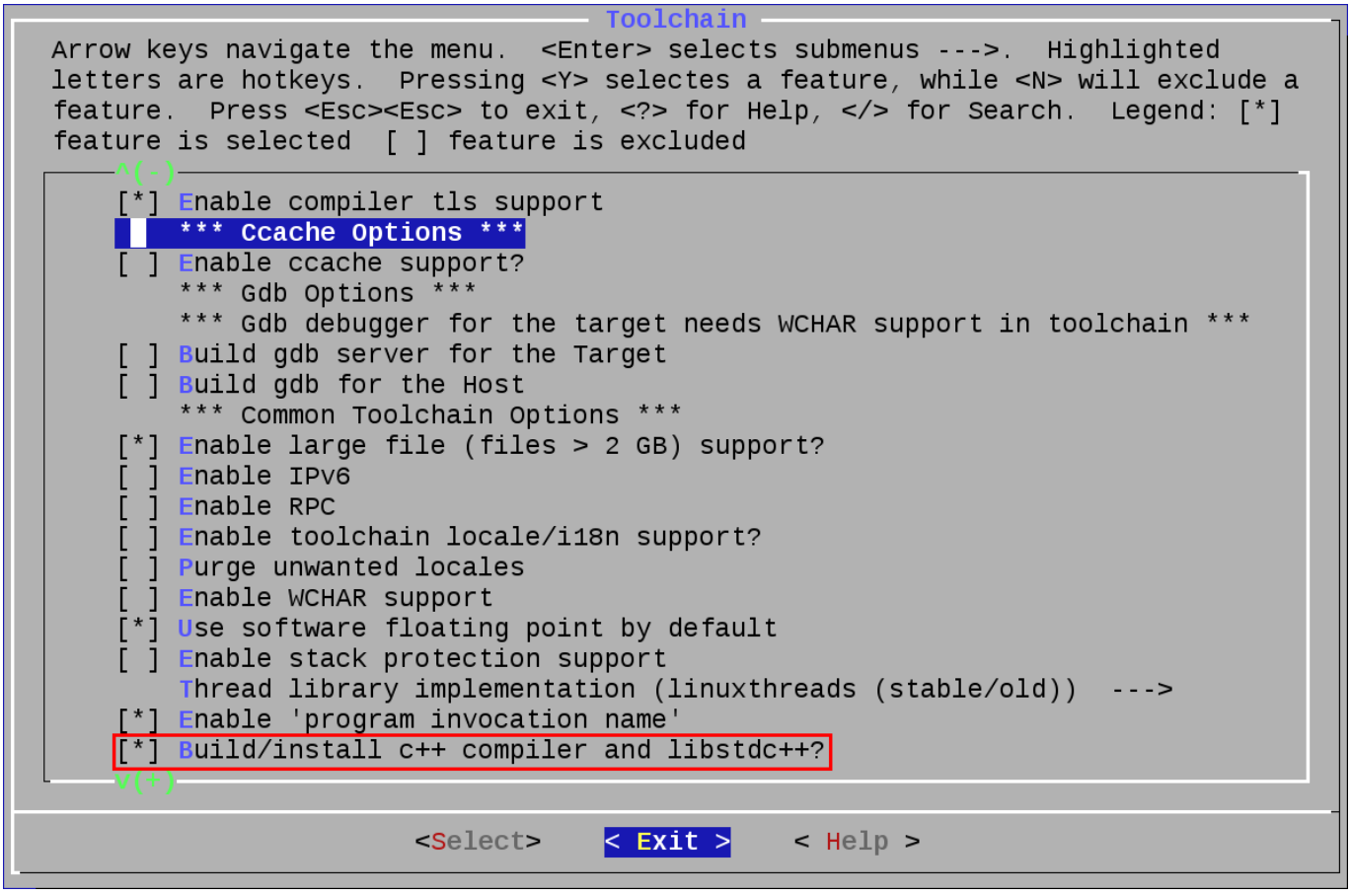
# Annexe 6 : Screenshot de menuconfig

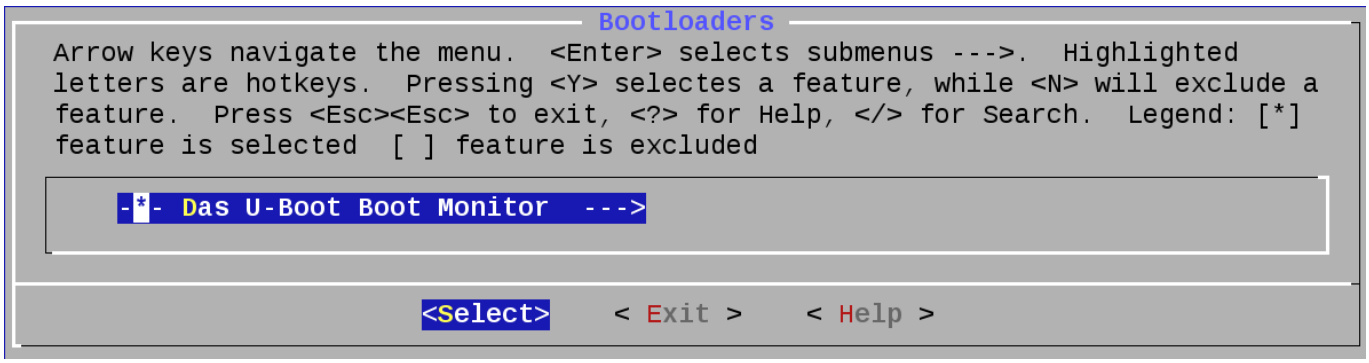
```
# make menuconfig
```

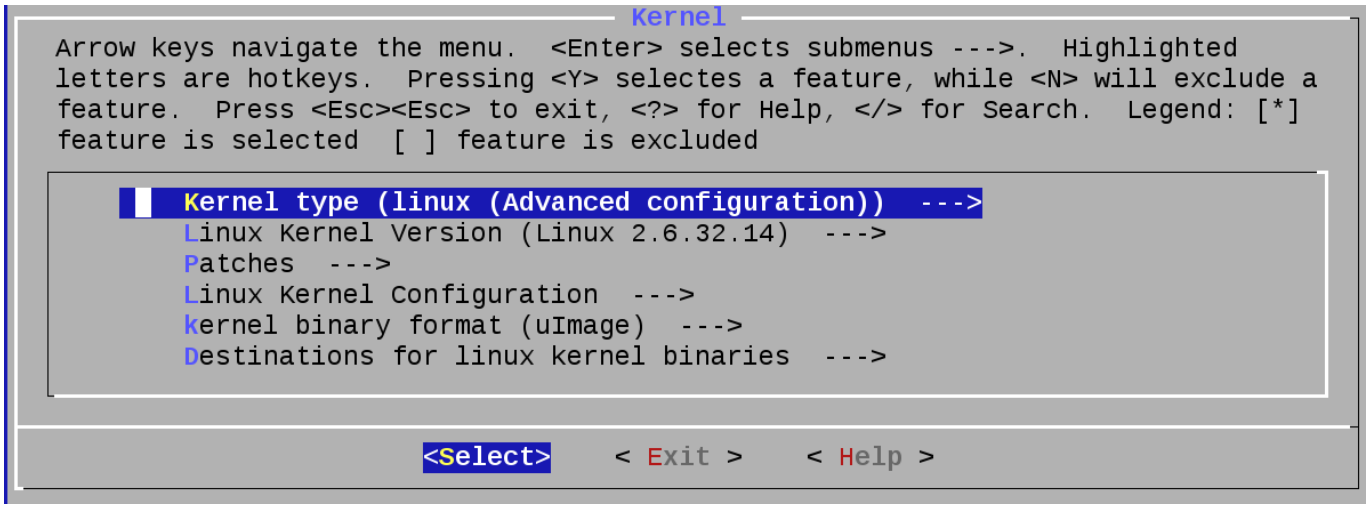



Toolchain
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected [ ] feature is excluded
Toolchain type (Buildroot toolchain) --->
*** Kernel Header Options ***
Kernel Headers (Linux 2.6.32.x kernel headers) --->
*** uClibc Options ***
uClibc C library Version (uClibc 0.9.31.x) --->
(toolchain/uClibc/uClibc-0.9.31.config) uClibc configuration file to use?
[ ] Thread library debugging
[ ] Compile and install uClibc tests
*** Binutils Options ***
Binutils Version (binutils 2.20.1) --->
(--with-arch=armv4t --with-cpu=arm9tdmi) Additional binutils options
*** GCC Options ***
GCC compiler Version (gcc 4.3.4) --->
[*] Enable toolchain with --sysroot support
[ ] Enable setjmp/longjmp exceptions?
() Additional gcc options
[ ] Fortran cross-compiler support
[ ] Build/install java compiler and libgcj?
[ ] Build/install Objective-C compiler and runtime?
v(+)
<Select> < Exit > < Help >


Toolchain
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected [ ] feature is excluded
^(-)
[*] Enable compiler tls support
*** Ccache Options ***
[ ] Enable ccache support?
*** Gdb Options ***
*** Gdb debugger for the target needs WCHAR support in toolchain ***
[ ] Build gdb server for the Target
[ ] Build gdb for the Host
*** Common Toolchain Options ***
[*] Enable large file (files > 2 GB) support?
[ ] Enable IPv6
[ ] Enable RPC
[ ] Enable toolchain locale/i18n support?
[ ] Purge unwanted locales
[ ] Enable WCHAR support
[*] Use software floating point by default
[ ] Enable stack protection support
Thread library implementation (linuxthreads (stable/old)) --->
[*] Enable 'program invocation name'
[*] Build/install c++ compiler and libstdc++?
v(+)
<Select> < Exit > < Help >


Bootloaders
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a
feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*]
feature is selected [ ] feature is excluded
-*- Das U-Boot Boot Monitor --->
<Select> < Exit > < Help >


Kernel
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a
feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*]
feature is selected [ ] feature is excluded
Kernel type (linux (Advanced configuration)) --->
Linux Kernel Version (Linux 2.6.32.14) --->
Patches --->
Linux Kernel Configuration --->
kernel binary format (uImage) --->
Destinations for linux kernel binaries --->
<Select> < Exit > < Help >

# Annexe 7 : Principales options de configuration

```
# vim .config
BR2_HAVE_DOT_CONFIG=y
BR2_VERSION="2010.05"
BR2_arm=y
BR2_arm920t=y
BR2_ARM_TYPE="ARM920T"
BR2_ARM_EABI=y
BR2_ARCH="arm"
BR2_ENDIAN="LITTLE"
BR2_GCC_TARGET_TUNE="arm920t"
BR2_GCC_TARGET_ARCH="armv4t"
BR2_GCC_TARGET_ABI="aapcs-linux"
BR2_GCC_TARGET_CPU="arm920t"
BR2_BOARD_NAME="phidget_sbc"
BR2_BOARD_PATH="target/device/Phidgets/phidgetsbc"
BR2_KERNEL_ARCH_PATCH_DIR="$(BR2_BOARD_PATH)/kernel-patches-$(BR2_LINUX26_VERSION
)"
BR2_TARGET_PHIDGETS=y
BR2_TARGET_PHIDGET_SBC=y
# BR2_TARGET_PHIDGET_SBC2 is not set
BR2_PHIDGETSBC_VERSION="1.0.4"
BR2_PHIDGETSBC_RELEASE_TYPE="minimal"
# BR2_PHIDGETSBC_RELEASE_FULL is not set
BR2_PHIDGETSBC_RELEASE_MINIMAL=y
BR2_TARGET_PHIDGET_SBC_BIN=y
BR2_TARGET_GENERIC_HOSTNAME="phidgetsbc"
BR2_TARGET_GENERIC_ISSUE="Welcome to Phidget SBC."
BR2_KERNEL_HEADERS_2_6_32=y
BR2_DEFAULT_KERNEL_HEADERS="2.6.32.14"
BR2_UCLIBC_VERSION_0_9_31=y
BR2_UCLIBC_VERSION_STRING="0.9.31"
BR2_BINUTILS_VERSION_2_20_1=y
BR2_BINUTILS_VERSION="2.20.1"
BR2_EXTRA_BINUTILS_CONFIG_OPTIONS="--with-arch=armv4t --with-cpu=arm9tdmi"
BR2_GCC_VERSION_4_3_4=y
BR2_GCC_SUPPORTS_SYSROOT=y
BR2_GCC_SUPPORTS_FINEGRAINEDMTUNE=y
BR2_GCC_VERSION="4.3.4"
BR2_TOOLCHAIN_SYSROOT=y
BR2_EXTRA_GCC_CONFIG_OPTIONS=""
BR2_GCC_SHARED_LIBGCC=y
BR2_GCC_ENABLE_TLS=y
...
BR2_GCC_CROSS_CXX=y
BR2_INSTALL_LIBSTDCPP=y
...
BR2_PACKAGE_BUSYBOX=y
BR2_BUSYBOX_VERSION_1_16_X=y
BR2_BUSYBOX_VERSION="1.16.1"
…
```

# Annexe 8 : Chien de garde (watchdog)

Un chien de garde (watchdog), est une technique logicielle utilisée pour s'assurer qu'un programme ne reste pas bloqué à une étape particulière du traitement qu'il effectue. C'est une protection destinée généralement à redémarrer le système, si une action définie n'est pas exécutée dans un délai imparti.

Il s'agit en général d'un compteur qui est régulièrement remis à zéro. Si le compteur dépasse une valeur donnée (timeout) alors on procède à un reset (redémarrage) du système. Si une routine entre dans une boucle infinie, le compteur du chien de garde ne sera plus remis à zéro et un reset est ordonné.

C'est une fonctionnalité primordiale des systèmes embarqués. La carte Phidget dispose d'un chien de garde.

## Le Watchdog est-il activé sur la phidget ?

Pour vérifier la présence active du watchdog, il suffit d'ouvrir une session ssh et de taper la commande :

```
# ps | grep watchdog
  836 root     /sbin/watchdog -t 5 -T 30 /dev/watchdog
```

Le processus watchdog s'exécute bien par défaut sur le système. Le timeout est ici réglé à 5 secondes. A l'expiration de ce timeout si le système ne répond plus, la carte sera automatiquement redémarré (reset).

Normalement, le watchdog est lancé au démarrage de la carte par le script :

```
# ls /etc/init.d/*watchdog
/etc/init.d/S98watchdog
```

```
# watchdog --help
BusyBox v1.16.1 (2011-02-04 12:37:16 MST) multi-call binary.

Usage: watchdog [-t N[ms]] [-T N[ms]] [-F] DEV

Periodically write to watchdog device DEV

Options:
        -T N    Reboot after N seconds if not reset (default 60)
        -t N    Reset every N seconds (default 30)
        -F      Run in foreground

Use 500ms to specify period in milliseconds
```

### *Comment activer le Watchdog sur la phidget ?*

Afin d'activer le watchdog, il faut simplement exécuter la commande suivante :

```
# /sbin/watchdog –t 5 -T 30 /dev/watchdog
ou
# /etc/init.d/S98watchdog start
```

Ceci permet d'activer automatiquement la fonction de watchdog sur la carte (dont un driver est en charge au travers de l'interface /dev/watchdog) et d'envoyer un « tic » périodiquement avant que le driver de watchdog provoque un reset la carte.

Afin de simuler (tester) cette situation, on peut terminer le processus /sbin/watchdog à l'aide de la fonction kill. La carte redémarrera 5 secondes après l'arrêt du programme.